LOADSTAR LETTER #62

Loadstar Begins a Commodore Home Page Service

By Jeff Jones. Having a professional design your web page can cost hundreds to thousands of dollars Many people try to do their own web page and never get it off the ground, or if they do, it, the results may not be pleasant. For only $25 per year, you can finally have your own finished, polished web page! Just mail or Email any amount of text about yourself. Include your life story, favorite recipes, even your portrait, drawings or baby pictures. Without any mental effort on your part, and very little money out of your pocket, It will be placed on the web with the address of

http://www.loadstar.com/yoursite/

Your text can be biographical, business oriented or editorial. You can also start an informational web page. Perhaps you want to spread the word about your favorite topic. Adding graphics (GIF or JPEG), supplied on disk, is $5 extra. If graphics must be scanned, add $10. Remember that the editor who brings your web page to life will be paid an hourly wage, so adding an hour to the job really does cost Loadstar more. So if your page will have many downloads, remember that these links be programmed and tested.

The best thing is you don't even have to be on the web in order to have a web page! Just write your text, preferably with TWS, SpeedScript or Edstar, and it will be incorporated into your page. You may also use PC-based word processors. We want to build a great online Commodore community.

The number of pages in your website depends on how much text you send. 200K of text, which is way more than most any website will have, is fine. If you want to move large archives to Loadstar's area, there may be an additional monthly charge per megabyte.

Maintenance of your website is also low-priced. Small changes which don't require completely revamping the site are $10. 00 and of course a complete overhaul is only $25. More details available online soon at our overhauled website.

jeff@LOADSTAR.com
US MAIL: ATTN. Jeff Jones
J & F Publishing
P.O. Box 30008 Shreveport
LA 71130-0008
Phone: 318/221-8718,
Fax: 318/221-8870

LOADSTAR Website To Become The Largest Commodore Knowledge base Ever

By Jeff Jones. By January 1st 1999 all the text ever published on Loadstar will soon be available online in a searchable format. This means that you can type a few keywords and in moments search though over a decade of knowledge. Every tutorial, every question and answer, every game, hint, and (God, no) every editorial will be there. It's like having nearly 200 issues of Loadstar at your fingertips. Plus, you can read the full text from any back issue before you purchase.

The Transactor Online Archive

http://csbruce. ml. org/~csbruce/commodore/transactor/

By Craig Bruce. This page is an online archive of The Transactor magazine. This archive is made available to the public with the written permission of the effective copyright holder of The Transactor, namely Karl Hildon, the Editor-in-Chief. Only a single issue is currently presented here, to "test the waters" of public interest in this project.

The Transactor was a popular magazine for the old 8-bit Commodore computers, which was renowned for its overall high quality and the unusually deep technical level of its articles. It was also produced in Canada. ;-)

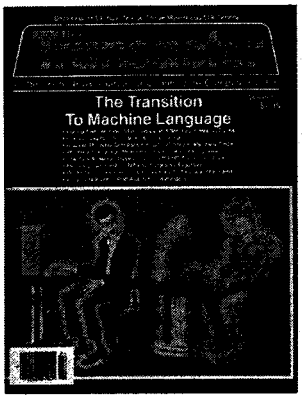


Image downloaded from archive.

The issue is presented in the form of large JPEG and GIF images. The magazine was printed with black & white pages inside of a color jacket (with cool art work), so the color jacket is in JPEG format and the inside pages are in 8-gray-level GIF format.

All pages are scanned at 120-dpi resolution so that they are readable and will fit across a 1024-pixel screen. The images are not (yet) available in a Commodore-readable format.

The page is actually "Lynx Friendly", although you will need some off-line means of viewing large GIF and JPEG images.

The available issue is Volume 5, Issue 02 (from 1984???) -- "The Transition to Machine Language".

Do check it out!

Fender Heads Out To Commodore Expo In Lansing Illinois

Excerpts from Fender's Discovery #172. To read more, check out LOADSTAR #172. I just returned from driving up to Lansing IL (a suburb of Chicago) for SWRAP's Chicago Expo 1998. There I hobnobbed with the world's 8-bit royalty -- Jim Butterfield, Maurice Randall, Dale Sidebottom, Scott Parker, Bob Bernardo, Robin Harbron, and many others. I figured that I would be the most traveled person there -- it was about a 900-mile trip -- but Bob Bernardo came from California and Robin Harbron and his crew came from Thunder Bay, Ontario. Jim Butterfield drove from Toronto.

It was a well-organized show put on by SWRAP officers Randy Harris, Eric Kudzin, Carol Bouslog, Glenn Knibbs,

Craig Prendota and other members of the club. It took place in a Holiday Inn meeting hall with tables covered with C-64/128 systems, some of them custom-made. The atmosphere was that of a "get-together" rather than a sell-fest. A few vendors were selling some used hardware and software but mostly everyone gabbed and showed off their new items. A lot of donated equipment (including some Cap'n Calhoon T-shirts) were given away in drawings throughout the day.

Dale Sidebottom, the major domo of the L.U.C.K.Y. club of Louisville KY, had a table set up with an accelerated 128 and a color laser printer. If I were any kind of reporter I would have written down the specifications of the magical stuff he was doing with his C-128, but instead I ended up with some color pictures of myself and others printed in 16 million colors. All of this PostScript, J-PEG, digital camera stuff is impossible to do with a Commodore -- according to every techie salesperson in the world -- but Dale did it. He took my picture with his digital camera and within two or three minutes I had the printout and a disk with all of the picture data on it...all done with his C-128.

Again, Robin will fill you in on the details in the next LOADSTAR Letter(s). By the way, Robin was quite impressive in the way he talked shop with just about everyone there. Some conversations would go way over my head but Robin seemed to know about every arcane topic mentioned. We are lucky to have knowledgeable and enthusiastic guys like Robin to write for us.

Randy also had his *supercharged SX-64* on hand for display. He and Eric Kudzin modified it by removing the 1541 and installing a CMD Hard Drive and FD-2000 internally! Add to that a CMD SuperCPU, a SmartTrack trackball, a PostScript Laser Printer, and WHEELS. Needless to say, this SX burns rubber!

http://members. aol. com/swrapug/expo98/expo98. htm

Dan's Web Tips: The Final Slash

By Dan Tobias. Always include the final slash (/) at the end of a URL that ends in a directory name. If you use:

```
http://www. someplace.
net/~msmith
```

(without the slash), the browser will first try to retrieve a file rather than a directory, and only when the server realizes that **~msmith** is a directory name will it tell the browser to add the slash and try again. This takes one extra communication round between browser and server, slowing down the retrieval. Also, the browser doesn't know in advance that the address without the slash goes to the same page as the one with it, so it won't show the link in the "visited-link" color if the user already went there, and won't take advantage of a previously-cached copy of the page that may exist.

Even worse, there are a few old browsers (some versions of Mosaic, for instance) that don't handle this sort of redirection correctly. They may pull up the correct web page without the slash, but they then fail to handle relative links from the page correctly. A link to **stuff. html** from the URL **http://www. someplace. net/~msmith/** should end up going to **http://www. someplace. net/~msmith/stuff. html**, but if the slash is omitted and the browser software isn't smart enough to add it once it's redirected by the server, it will think it's really one directory level higher in the tree, and parse the relative URL as **http://www. someplace. net/stuff. html**. This will then cause a 404 Not Found error, and the user won't know why.

One very prominent site whose creators failed to heed this advice is the official government posting of the Ken Starr Report on President Clinton's relations with intern Monica Lewinsky. Due to news-media hype, this report (posted to several official government sites on September 11, 1998, and shortly thereafter to various private-sector sites as well) got some of the heaviest Internet traffic ever, causing the servers to be so overloaded in the first few hours the report was up that most people couldn't connect. Unfortunately, the government added to this problem by using versions of the URLs of these sites lacking the trailing slash everywhere they publicized or linked to the sites, thus ensuring that each access of the site would have one more server transaction than would be necessary if the slash had been used. With the high level of traffic the site had at the time, this probably added long delays for many people's accesses.

Having said all this, I'd better remind you **not** to "overcorrect" by adding slashes to URLs that aren't supposed to have them. If the URL references a *file* rather than a *directory*, there shouldn't be a slash at the end. So **don't** type "`http://www. someplace.net/~msmith/stuff.html/`"!

Dan breathes HTML for Softdisk. You can find Dan's Web Tips at http:// www. softdisk. com/comp/dan/webtips/

Top Down Programming With Mr. Mouse

By Jeffrey L. Jones. When I first dabbled in programming, ON GOTO confused me and I avoided it. I even skipped over reading about it. I had actually never used ON GOTO to its fullest until I began working at LOADSTAR. With Mr. Mouse, you'll be receiving numbers for menu items, regions, etc. In BASIC, the best way to handle numeric input from a program is with the ON command

ON GOSUB is simple and powerful in nature. That makes it a joy to program with because once you've

A dual column menu returning the numbers 1-12

gotten to the ON GOSUB line, your program REALLY starts moving along.

ON GOSUB is kind of like a menu for BASIC. Based on a numeric evaluation, you're giving BASIC a choice of line numbers to jump to:

```
50 geta$:ifa$<"1"ora$>"4"then50
60 onval(a$)gosub80,90,400,20:goto50
```

On line 60 the program would jump to:

line 80 if a equals 1
90 if a equals 2

400 if a equals 3
20 if a equals 4.

After RETURNing from the subroutines, or if somehow a isn't in the range of 1-4, the program will pick up on line 60 and goto line 50.

For those not familiar, the GOSUB command is just like GOTO, but it remembers where it came from. The RETURN command is paired with GOSUB.

If you're anything like me, you're confused now. Let me put it another way:

BASIC sees line 60 as a MENU, or option table. It has four choices, 1, 2, 3 and 4. After the ON GOSUB command, BASIC will count the items in the menu of line numbers to GOSUB.

Here are BASIC's options:

You've already limited input to the numbers 1-4. You can't get past line 50 without pressing one of these keys.

For BASIC, line 80 is option 1, 90 is option 2 and so on. The list can go on as long as your 80 character BASIC line limit allows.

The command, "goto 50", will be executed if val(a$) is zero or val(a$) is greater than the number of line numbers in the list. In other words, ON GOSUB will fall through to the next command or line. Of course after BASIC RETURNS from the subroutines, it will goto 50 also.

So you have some automatic logic built into ON GOTO/GOSUB. The code above would work exactly the same without limiting your input:

```
 50 geta$:ifa$=""then5
60 onval(a$)gosub80,90,400,20:goto50
```

There is no need to test a$ to see if it's out of range. ON GOSUB will not execute and move on to the next command (or line) if a$ is zero or greater than the number of items on the list.

Here is the equivalent without the ON GOSUB combination:

```
 50 geta$:ifa$<"1"ora$>"4"then50
55 ifa$="1"thengosub80
60 ifa$="2"thengosub90
65 ifa$="3"thengosub400
70 ifa$="4"thengosub20
75 goto50
```

Now that we have that out of the way, we can move on to real menus. A real menu is a part of a program that offers the user a choice. Undoubtedly you've bumped into menus before.

There are one-key menus where you're asked to press a single key in order to invoke an action. Then there are CRSR-RETURN menus that allow you to move a highlight bar to an item and then press RETURN to select it. Sometimes there can be a combination of the two, meaning you can use CRSR/ RETURN or press one of the numbered options, your choice. With Mr. Mouse, you are offered simple menus that use CRSR/RETURN and the mouse, multi-column menus as pictured previously, and scrolling menus.

For a one key menu, it's pretty obvious that you PRINT the menu to the screen in whatever way you feel comfortable with, then wait for input.

Here's a sample one-key menu:

```
1. Enter Data
2. Load Data
3. Save data
4. Start over
 [Press a Key, 1-4]
```

How you actually print your menu is up to your own artistic taste. Next you will have to wait for input. The most common way is to GET a string:

```
50 geta$:ifa$=""then50
60 onval(a$)gosub100,200,300,400
70 goto50
```

With Mr. Mouse you would set up a menu with hotkeys defined as 1-4. So the user can click on either line or press 1-4 or cursor to any line and press return. Your program couldn't be friendlier. Some people might think that's overkill, but it's really not. These days it's expected. When I look at a screen, I ask myself "What might the user *try*? Is it *reasonable* to expect my program to respond to such input?" If so, I try to implement the input, even if I never mention it in the docs or on the screen. With the mouse pointer active, the user might just click on anything.

TOP DOWN

You're about to start on an application program. It's a phone list manager. You type in the first line:

```
 10 poke53280,0:poke53281,0:print
[CLR]
```

Okay, you know what color the border and background are going to be but what about the program itself? How will you create a usable program from what is essentially a thought? What I'm about to say might go against some people's natural urges but the "top-down" approach is usually the best (as well as fastest) way to create programs.

Top-down means designing or creating the interface (menus) first, then creating the guts of the program. You print the menu on the screen, even have a working highlight bar. If an item on the menu should have a sub-menu, get that working, too. So essentially you start off with a program that looks finished, shows you most of its functions, but doesn't work. In my book that's one level better than starting off with a program that DOESN'T look finished and doesn't work to boot. This gives you an animated outline of the finished product. With the main menu working, at least you can visualize how the program will work and not leave anything out.

Imagine building the elevators and cafeteria before the building. At least if you wait till the building is erected and floors assigned, you'll know how many buttons to fit into the elevators, and which floors should be skipped. The old horse before the cart syndrome is similar to creating a program before creating its interface with the user.

Filling in your hollow program becomes a breeze. All you do is "fill in" the features, starting from the top of your menu and working your way down. Programming this way, you'll discover that the program almost writes itself for you. Additionally your code will automatically become sectioned and easily modifiable. You're sub- routining and even writing structured code without having to think about it. You may also find that in the process of filling in features, you're creating routines that other features can use to get their jobs done.

PITFALLS TO SUBROUTINING

While on the subject of subroutining, three commonly wasteful practices used by many programmers are:

a. Clearing lines on the screen
b. Plotting the cursor
c. Overcoming line link problems

Probably no one sees as much of

Craig Prendota and other members of the club. It took place in a Holiday Inn meeting hall with tables covered with C-64/128 systems, some of them custom-made. The atmosphere was that of a "get-together" rather than a sell-fest. A few vendors were selling some used hardware and software but mostly everyone gabbed and showed off their new items. A lot of donated equipment (including some Cap'n Calhoon T-shirts) were given away in drawings throughout the day.

Dale Sidebottom, the major domo of the L.U.C.K.Y. club of Louisville KY, had a table set up with an accelerated 128 and a color laser printer. If I were any kind of reporter I would have written down the specifications of the magical stuff he was doing with his C-128, but instead I ended up with some color pictures of myself and others printed in 16 million colors. All of this PostScript, J-PEG, digital camera stuff is impossible to do with a Commodore -- according to every techie salesperson in the world -- but Dale did it. He took my picture with his digital camera and within two or three minutes I had the printout and a disk with all of the picture data on it...all done with his C-128.

Again, Robin will fill you in on the details in the next LOADSTAR Letter(s). By the way, Robin was quite impressive in the way he talked shop with just about everyone there. Some conversations would go way over my head but Robin seemed to know about every arcane topic mentioned. We are lucky to have knowledgeable and enthusiastic guys like Robin to write for us.

Randy also had his *supercharged SX-64* on hand for display. He and Eric Kudzin modified it by removing the 1541 and installing a CMD Hard Drive and FD-2000 internally! Add to that a CMD SuperCPU, a SmartTrack trackball, a PostScript Laser Printer, and WHEELS. Needless to say, this SX burns rubber!

http://members. aol. com/swrapug/expo98/expo98. htm

Dan's Web Tips: The Final Slash

By Dan Tobias. Always include the final slash (/) at the end of a URL that ends in a directory name. If you use: `http://www. someplace. net/~msmith` (without the slash), the browser will first try to retrieve a file rather than a directory, and only when the server realizes that **~msmith** is a directory name will it tell the browser to add the slash and try again. This takes one extra communication round between browser and server, slowing down the retrieval. Also, the browser doesn't know in advance that the address without the slash goes to the same page as the one with it, so it won't show the link in the "visited-link" color if the user already went there, and won't take advantage of a previously-cached copy of the page that may exist.

Even worse, there are a few old browsers (some versions of Mosaic, for instance) that don't handle this sort of redirection correctly. They may pull up the correct web page without the slash, but they then fail to handle relative links from the page correctly. A link to **stuff. html** from the URL **http://www. someplace. net/~msmith/** should end up going to **http://www. someplace. net/~msmith/stuff. html**, but if the slash is omitted and the browser software isn't smart enough to add it once it's redirected by the server, it will think it's really one directory level higher in the tree, and parse the relative URL as **http://www. someplace. net/stuff. html**. This will then cause a 404 Not Found error, and the user won't know why.

One very prominent site whose creators failed to heed this advice is the official government posting of the Ken Starr Report on President Clinton's relations with intern Monica Lewinsky. Due to news-media hype, this report (posted to several official government sites on September 11, 1998, and shortly thereafter to various private-sector sites as well) got some of the heaviest Internet traffic ever, causing the servers to be so overloaded in the first few hours the report was up that most people couldn't connect. Unfortunately, the government added to this problem by using versions of the URLs of these sites lacking the trailing slash everywhere they publicized or linked to the sites, thus ensuring that each access of the site would have one more server transaction than would be necessary if the slash had been used. With the high level of traffic the site had at the time, this probably added long delays for many people's accesses.

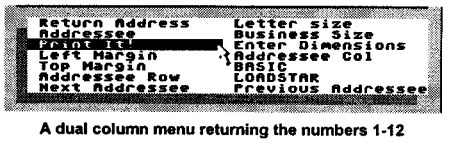
Having said all this, I'd better remind you **not** to "overcorrect" by adding slashes to URLs that aren't supposed to have them. If the URL references a *file* rather than a *directory*, there shouldn't be a slash at the end. So **don't** type "`http://www. someplace.net/~msmith/stuff.html/`"!

Dan breathes HTML for Softdisk. You can find Dan's Web Tips at http://www. softdisk. com/comp/dan/webtips/

Top Down Programming With Mr. Mouse

By Jeffrey L. Jones. When I first dabbled in programming, ON GOTO confused me and I avoided it. I even skipped over reading about it. I had actually never used ON GOTO to its fullest until I began working at LOADSTAR. With Mr. Mouse, you'll be receiving numbers for menu items, regions, etc. In BASIC, the best way to handle numeric input from a program is with the ON command

ON GOSUB is simple and powerful in nature. That makes it a joy to program with because once you've



A dual column menu returning the numbers 1-12

gotten to the ON GOSUB line, your program REALLY starts moving along.

ON GOSUB is kind of like a menu for BASIC. Based on a numeric evaluation, you're giving BASIC a choice of line numbers to jump to:

```
50 geta$:ifa$<"1"ora$>"4"then50
60 onval(a$)gosub80,90,400,20:goto50
```

On line 60 the program would jump to:

line 80 if a equals 1
90 if a equals 2

horizontal lines? These lines are called scan lines. At the back of the picture tube is an electron gun. It is like a machine gun, continually firing electrons at the screen. The back of the screen is coated with phosphorus, and whenever an electron hits it, it flashes. The gun sweeps back and forth across the screen from top to bottom. It moves with such incredible speed that before the flash dies out in one spot another electron will hit it. To the slow human eye, all this appears to be a solid screen. If you're wondering how fast the raster line is, it's approximately 1. 92 MILLION pixels per second with each raster line being drawn with $D011-$D012 updated every 1/6000th sec. So you see that ML is fast enough to monitor the raster line, but you can't add much to the code before the overhead makes it miss a few clicks of the line.

The raster is the current scan line that the gun is shooting at. The value of the raster will vary from 0 - 262 on normal TVs, on a European TV it goes from 0 - 312. Since these values are greater than 255 we can't fit them all in one byte, therefore 9 bits are used to store the raster. The lower 8 bits are stored in $D012 and the 9th bit is stored in bit 7 of $D011. Note that the top of the screen corresponds with a value of 50 and the bottom is at 249. The raster value changes so fast that even a machine language program can't check it fast enough if it's going to do anything else. Fortunately, the hardware can check it for us. We can tell it to trigger an IRQ every time the raster equals a specific value. This is how you write split-screen applications.

For a screen half high-res and half text, set the screen to high-res mode and tell it to interrupt at half way down the screen. Using a wedge we can catch it when this happens and set the screen back to text. By telling it to interrupt when we get back to the top we can change back to high-res. By flipping back and forth like this we achieve the desired effect. Our program doesn't even have to check the raster at all.

The VIC chip is capable of creating four types of interrupts. They are:

raster interrupt (bit 0)
sprite/foreground collision (bit 1)
sprite/sprite collision (bit 2)
light pen interrupt (bit 3)

These interrupts do not necessarily occur every 60th/sec. They happen when a predetermined condition is met, which means they don't eat up any cycles until you want them to. Even then RASTER routines are generally VERY short and efficient. You'll see why later.

The initialization of a raster interrupt is a bit more involved than what we've seen so far. A few more locations come into play. These are:

$D011-12 - Current raster line
$D019 - Interrupts status (INSTAT)
$D01A - IRQ mask register
$DC0D - interrupt control register

Like everything else, interrupts have special memory locations. Logically, the first thing we need is a way to tell exactly what caused the interrupt. The story is told in the INTERRUPT STATUS REGISTER, which is called INTSTAT for short. Don't get this mixed up with the normal status register, SR. The INTSTAT is stored in $D019 or 53273. Each bit of the INTSTAT refers to a different interrupt.

NAME	BIT	CAUSE OF INTERRUPT
IRST	0	Raster
IMDC	1	Sprite-Background
IMMC	2	Sprite-Sprite
ILP	3	Light pen

Note: the 7th bit is the IRQ and this bit is set on ALL interrupts.

GETTING DOWN TO IT

First we have to tell it on what line to interrupt. You do this by storing the value in the raster $D012 and bit 7 of $D011. You CANNOT just ignore this extra bit!

NOTE: This is also where the current raster is stored. You probably wonder how it can hold both values at once. Remember we are dealing with an IA register here not real memory.

Next, we must enable the raster interrupts. To do this, set bit 0 of the INTENAB. Finally, we must have our own interrupt wedge installed. It will have to determine whether this is a normal IRQ or from the raster. If normal then JMP to the normal vector. If it's raster then do whatever we want, clear the latch, reset where we want raster interrupts to occur, then restore the registers ourselves and return from the interrupt. Ready for the next program? This program divides the border in 2 parts depending on where the cursor is at using raster interrupts.

The SEI command provides an easy way of momentarily disabling ALL interrupts. However, sometimes we want just to turn off some of them. This can be done with the INTERRUPT ENABLE REGISTER, or INTENAB. It is at $D01A or 53274. Its format is identical to the INTSTAT, but it performs a totally different function. Care must be taken not to get them confused. Remember the INTSTAT shows what caused the interrupt. The INTENAB determines what CAN CAUSE an interrupt. To enable an interrupt from a source, its corresponding bit in INTENAB must be turned on. Use the same table above. For example, set bit 2 to enable Sprite-Sprite interrupts.

NOTE: The INTSTAT can still be checked even if the interrupt for that bit is disabled. If the condition is right, the INTSTAT will be set, but it won't trigger an IRQ (sprite collisions are a good example). Once again, you POKE the INTENAB to select which interrupts can occur and PEEK the INTSTAT to find what caused the interrupt.

Also, when you write your own routines to service interrupts, you must remember to clear the interrupt when you're finished. INTSTAT gets 'latched' which means, once a bit is set it stays set until cleared. To clear the interrupt you write a 1 to the corresponding bit in INTSTAT. This does seem backwards, but that's how it works. Therefore, after you get done handling a raster interrupt (bit 0):

```
LDA #$01:  STA INSTAT
```

This way if several bits are set you can handle them one at a time.

First comes the setup, with an explanation of each command.

```
SETUP SEI
      LDX $314
      LDY $315
      STX OLDIRQ
      STY OLDIRQ+1
      LDX #<NEWIRQ
      LDY #>NEWIRQ
      STX $0314
      STY $0315
```

We've seen this before. This is wedging in our routine into the interrupt vector. Nothing new here.

```
      LDA #$01
      STA $D01A
```

other people's code as we do. The aforementioned tasks commonly cause budding programmers to make the mistake of defining large strings over and over again, which eventually take their toll through garbage collection. Garbage collection causes your computer to pause while it cleans up the mess that the program has made of memory. Garbage collection is almost always the programmer's fault.

These pitfalls also take their toll by making programs larger. I'm not talking a little larger -- I'm talking about programs that we've published at 60-blocks that were 110 when we received them.

To clear a line on the screen, please DON'T define 40 spaces and print them. Instead:

```
POKE 781,line:SYS 59903
In assembly language:

LDX #line
 JSR $E9FF  ;59903
```

To clear a range of lines, do it in a FOR NEXT loop. Better yet, use the block routine in Mr. Mouse with sysml+30,x1,x2,y1,y2,screen code, color

To plot the cursor on a particular row and column, just do the following in BASIC:

```
POKE214,row:POKE211col:SYS
58732
```

That is the most efficient way for BASIC. In assembly language the KERNAL PLOT routine is better suited:

```
LDX #row
LDY #column
CLC
 JSR $FFF0
```

To TAB backwards:

```
POKE 211,column:SYS 58732
```

In assembly language:

```
LDA #column
 STA 211
 JSR 58732   ;$E56C
```

Of course with Mr. Mouse, you should use the Print at command, SYSML+12,x,y,string.

HOW TO BLOAD ML AND PROTECT BASIC

By Fender Tucker. A question often asked by beginning programmers is, "How can I use machine language routines in my BASIC program?" It's easy once you understand the principles.

An ML routine must be placed in memory somewhere. Usually it has to be placed EXACTLY where the ML programmer wrote it to be. There are many areas that can be used, with pages 192 through 207 the most popular. This is known as the $c000 area -- the "C thousand" area.

This is one way of loading an ML module into place:

```
 10 ifa=0thena=1:load"routine",8,1
```

Whenever BASIC does a load like this, it starts over at the beginning of the program. That's why this method includes the part

```
 ifa=0thena=1
```

This keeps the program from loading the ML over and over and over. . .

The following is the method we use and recommend for loading ML or any kind of binary file.

```
 10 sys57812"routine",8,0:poke780,0:
poke781,lb:poke782,hb:sys65493
```

Here's how you find hb and lb. The hb is the page number, pure and simple. The low byte is the byte on that page where the routine starts. Generally the routine will start at the beginning of a page so lb will be 0.

Let's say you have a routine that is called by SYS49152, according to the instructions. To find the page number, divide 49152 by 256. You get 192. To find the low byte, multiply the page number by 256 and subtract that from 49152. In other words, you are finding the remainder when 49152 is divided by 256. In this case, the remainder is 0. This means lb = 0 and the routine starts at the beginning of page 192.

Sometimes ML is written for locations other than the $c000 area. Sometimes memory is to be taken away from the BASIC program storage area. The area reserved for BASIC programs is from page 8 to page 160. You can change these numbers. To change the start of BASIC from page 8 to page 16 you'd do this:

```
poke16*256,0:poke44,16
```

About the only thing you can do after this, if it's in a program, is boot up another program. BASIC would get quite confused if your code is on page 8 but you tell the computer that BASIC is now starting at page 16. If you didn't load code there, or already have code there, you'd have a crash.

The poke16*256,0 is necessary because wherever BASIC is, it must start with a 0 byte.

To change the end of BASIC to page 152 you would do this:

```
 poke55,0:poke56,152:clr
```

This would allow you to use pages 152 through 159 for ML or other data. The beginning of BASIC change would allow you to use pages 8 through 15.

So, in a nutshell, here is what you need to have in your program in order to use someone else's ML routine. Let's say the routine is called with SYS49152.

```
 10 rem load the ml
 20 sys57812"routine",8,0:poke780,0:
poke781,0:poke782,192:sys65493
 100 sys49152:rem do the job
```

If the routine were called with SYS16384, then you'd:

```
 10 poke64*256,0:poke55,0:poke56,64:
clr:rem move top of basic down freeing
up pages 64 through 159
 20 sys57812"routine",8,0:poke780,0:
poke781,0:poke782,64:sys65493
 100 sys16384:rem do the job
```

Note that the number POKEd into 782 is the page number, or the SYS number divided by 256.

Raster Interrupts

By Steve Emsley & Roy Riggs with additional material by Jeff Jones. So far, we've looked at interrupts that are generated by the CIA 1 chip. A more respected type of interrupt is one generated through the VIC-II chip. These are the routines that let you create split screens, smooth scrolls, more than eight sprites/ more than one font on the screen at once, no borders, and many, many more possible effects.

Raster interrupts are kind of tricky, and most programmers are still in the dark about them. Let's cover the basics here. First, the picture generated by your TV or monitor is NOT projected all at once. Look closely at your screen now, notice how it's made of a bunch of

```
    RTS
```

At this point, our routine is wedged into the IRQ vector, and a raster interrupt will be triggered when the scan line hits line 48.

ADVANCED WEDGE

For our next routine, let's split the screen into three modes: multicolor, text and hires. The multicolor will be from line 48 (top of the screen) to about line 129. The regular text screen will pick up at line 130 ($82), and continue to line 177. The hires will go from line 178 ($B2) to line 255, or off the screen.

The first thing we need to do in our routine is let the computer know the type of interrupt. This is done by turning on the appropriate bit in location $D019. Since we want a value of 1, this can be done a number of ways.

```
NEWIRQ LDA #$01
    STA $D019
or:
    INC $ D019
or:
    ROLL $ D019
or:
    LDA $ D019
    STA $ D019
```

Now to get the current raster line.

```
    LDA $ D012
    CMP #$30  ;is it 48?
    BEQ MULTI
    CMP #$82  ;is it 130?
    BEQ TEXT
```

At this point, if it's not 48 or 130, it must be 178. Since the routine is triggered by our writing to $D012, it will only go into effect when its value is 48, 130 or 178. We'll label this part HIRES. Even though this is the 'end' of the routine, it's going first. Of course, you could always have a BNE HIRES, but that would be superfluous.

```
HIRES LDA #$3B
    STA $D011 ;53265,59
    LDA #$C8
    STA $D016 ;53270,200
    LDA #$30
    STA $D012
    JMP $EA31
```

The last STA is to prime the routine for the next interrupt sequence. This happens in a several millionths of a second so you know that the raster line hasn't moved there yet. Notice how we are going to the regular interrupt routine here.

```
MULTI LDA #$3B
    STA $D011 ;53265,59
    LDA #$D8
    STA $D016 ;53270,216
    LDA #$82
    STA $D012
    JMP $EA81
```

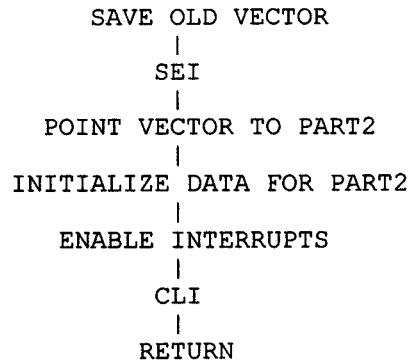
Here we put the computer into multicolor mode, and set up the next interrupt for line 130 ($82). Notice we jump to $EA81, which is the same as PLA TAY PLA TAX PLA RTI. This way all we're doing is returning from the interrupt.

```
TEXT  LDA #$1B
    STA $D011 ;53265,29
    LDA #$C8
    STA $D016 ;53270,200
    LDA #$B2
    STA $D012
    JMP $EA81
```

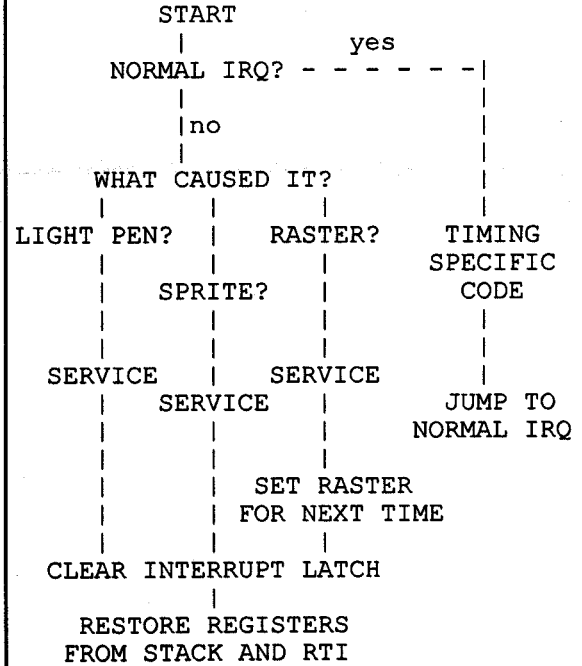
FLOWCHART

Below is a flow chart for both types of interrupts discussed. The flowcharts are the skeleton for any interrupt routine you may write. Leave out the branches you don't use and flush out the others to suit your purpose.

TIMER INTERRUPT



RASTER/OTHER INTERRUPTS



This location is where we 'more or less' tell the computer what type of VIC interrupt we want. Remember, a raster interrupt is bit 0, which is set by a value of 1.

```
LDA #$7F
STA $DC0D
```

This disables all interrupts. This might seem a bit repetitive, since we started with an SEI (which disables interrupts). The CIA# 1 has five interrupt sources, and we don't want those 'interrupting' us.

NOTE: If you're wondering what the five sources are, they are:

Timer A counting down to zero
Timer B counting down to zero
TOD clock reaching alarm time
Serial shift reg. compiled a byte
The FLAG line of the datasette

```
    LDA $D011
    AND #$7F
    STA $D011
    LDA #$30
    STA $D012
```

Location $D012 is the raster line, and has two functions, depending on whether it is read from or written to. Reading it (PEEKing) will ALWAYS tell the current raster line. Writing a number DESIGNATES the next desired line to interrupt. So, putting a $30 (48 decimal) will trigger a raster interrupt when the raster scan is on line 48, which is right where the top of the screen starts.

If we don't and $d011 with $7f (% 01111), we could be setting an interrupt for line 304 instead of line 48, which will never happen.

Now all that is left is to clear the interrupt and return. Here is how the initialization process looks when it's together.

```
SETUP SEI
    LDX #<NEWIRQ
    LDY #>NEWIRQ
    STX $0314
    STY $0315
    LDA #$01
    STA $D01A
    LDA #$7F
    STA $DC0D
    LDA $D011
    AND #$7F
    STA $D011
    LDA #$30
    STA $D012
    CLI
```

Glossary Of Commodore Terms

6510 - chip used as the CPU for a C-64.
8502 - chip used as the CPU for a C-128.
ASCII - American Standard Code for Information Interchange. This ensures that everyone uses the same number for a particular character.
BAM - Block Allocation Map - A track on a disk that keeps track of all of the files and their locations on the disk. On a 1541 disk, it's at Track 18, sector 0.
BANK - a portion of computer memory. For Commodores it means a 16K section of memory.
BASIC - acronym for Beginner's All- purpose Symbolic Instruction Code. The language of choice for C-64ers.
BAUD - the rate at which data is transferred, usually referring to modem speed. 1200 BAUD equals 1200 BITS per second or 150 BYTES per second.
BBS - Bulletin Board System - a program that allows a computer to be used as an electronic forum by modem users.
BIT - BInary digiT. The smallest unit of data available on a computer.
BOOT - generally, a short program that executes a longer one. If you see a file with the word "boot" in it on LOADSTAR, it means that this is the file to RUN to get the program going.
BYTE - 8 bits of data. One memory location.
CIA - Complex Interface Adapter – the chip that controls the joystick, keyboard and other items.
COMPOSITE - A video format where luma, chroma are combined into one signal. The typical signal used in your VCR's video in and video out plugs. Most 40-column monitor use composite video.
CPU - Central Processing Unit - the chip(s) that do the routing and bookkeeping in your computer.
CRSR - short for cursor, the pointer (often blinking) that shows you where you are on the screen.
CRT - Cathode Ray Tube - your monitor or TV set's picture tube.
DATABASE - any computer-usable collection of information or a program that helps you keep track of data, often allowing you to manipulate the data easily.
DEC - short for decimal, the number system we, as humans, use. Base 10.
DEFAULT - the mode that a program chooses for you before you choose your own mode.
DOS - Disk Operating System. In the MS-DOS world, DOS is short for MS-DOS or Microsoft Disk Operating System. In the old days PCs had very little ROM and had to LOAD the entire operating system from disk. C- 64/128s have always had the entire operating system in ROM. For Commodore users DOS means the programs that control your disk drive, which is a computer itself. It, too, has quite an elaborate operating system stored in ROM.
DOT MATRIX - Referring to printers, this is the practice of using dots to represent characters or graphics. Dot matrix offers speed and low-end costs while sacrificing the print- quality of DAISY WHEEL and LASER printers.
DRAFT MODE - the default, fastest mode for a printer.
F-KEYS - function keys - programmable keys that are separate from the regular QWERTY keyboard layout. Used for whatever the programmer wants. These are just "extra" keys that you can tell the user to push supposedly without seeming as cryptic as CTRL-X.
GUI - (gooey) Graphic User Interface such as the one used in GEOS
HEX - short for hexadecimal, the number system programmers prefer. Base 16.
K - Shorthand for KILO or 1024 for computers. 64K means 64*1024, or 65,536.
MEG - 1 million. A MEGABYTE is roughly one million bytes (actually 1K squared, 1024*1024 bytes or 1,048,000 bytes)
MEGAHERTZ measure of millions of cycles per second.
MENU-DRIVEN - the interface system LOADSTAR usually uses. You chose from a list of options by moving a highlight bar or pressing a key.
MIDI - Musical Instrument Digital Interface. This is the hardware standard for how electronic musical devices connect and talk to one another.
ML - machine language. The lowest, yet fastest, level at which to speak to your computer. Not for beginners.
MODEM - MOdulator/DEModulator - device used for transferring data from one computer to another.
MONOCHROME - a one-color monitor.
NLQ - Near Letter Quality - term for the slowest, yet best-looking printout on a dot matrix printer.
NYBBLE - Four BITs or half a BYTE
OCT - short for octal, another system that programmers like. Base 8.
PETASCII - Wouldn't you know it, Commodore doesn't use the same standard as everyone else. PET is an early name for Commodore computers. The main differences between PETASCII and ASCII are in the cases of the alphabet and in the graphics. Most punctuation marks are the same in either system.
PRG - short for PRoGram file. Used for storing data along with a memory address.
RAM - Random Access Memory - chips that can be written to and erased.
REL - RELative file. A file that can be accessed anywhere, and doesn't need to be read from the beginning.
REM - a REMarked line in a BASIC program. It's not executed, it's just for information.
RESET - standard on the C-128, a switch that allows you to restart your computer without wiping out everything in memory.
RGB - Red, Green, Blue separated video signals, typically used 80- column monitors.
ROM - Read Only Memory - chips that are "burned" in and cannot be changed by the user.
SCSI - Small Computer System Interface, typically used in your better hard drives. Just a standard of connectivity between devices, just like MIDI.
SECTOR - a portion of a track, often called a "block". There are 664 of them available on a blank diskette, not including the sectors used in the directory. Each one can hold 254 bytes of data and 2 bytes for DOS information.
SEQ - SEQuential file. Used for storing data without an address.
SID - Sound Interface Device - the chip that supplies sound on Commodore computers.
SPREADSHEET - a program that allows you to enter rows and columns of information and make fast calculations on them.
TRACK - a concentric division of the actual disk. There are usually 35 tracks on a 1541 disk.
USR - same as SEQ. Meant to be more personal for USeRs than SEQ.
VDC - Video Display Chip - controls the 80-column screen.
VIC - Video Interface Chip - controls the 40-column screen.
WIMP - Window-Icon-Mouse-Pointer - an interface system used by Macintoshes, new IBMs, GEOS, etc.
WYSIWYG - "What You See IS What You Get" - usually means that a printout will look EXACTLY like what you see on the screen.
WORD PROCESSOR - a program that takes the place of a typewriter allowing you to organize the text before it's printed out.

```
CHKIN     $FFC6  65478
CHKOUT    $FFC9  65481
READSST   $FFB7  65463
CLOSE     $FFC3  65475
```

GET'NUMBER AND GET'STRING

We'll be incorporating GET'NUMBER and GET'STRING quite a bit from now on.

GET'NUMBER is an ML routine that reads numeric parameters from SYS commands like the following:

SYS 49152,12

You JSR to GET'NUMBER when your ML is ready for the data. The first thing you do does NOT have to be a JSR to this routine. After a JSR to GET'NUMBER, the .Y register will hold the LSB of the parameter and the .A register will hold the MSB.

GET'NUMBER will accommodate INTEGERS from 0-65535. Any fractional part will be ignored. The number can be a literal, variable or formula, limited by the parenthetical formula restraints of BASIC. There must be one JSR to GET'NUMBER for every numeric parameter on the SYS.

GET'STRING is an ML subroutine that will process strings on a SYS command like the following:

SYS49152,"text"

You JSR to GET'STRING when your ML is ready to process the string. After JSRing to GET'STRING, $22-$23 will point to the string. For good measure, . X and . Y will also point to the string. Thus, if the string is a filename, you're ready to JSR SETNAM immediately after a JSR to GET'STRING.

GET'STRING will accommodate strings up to 255 characters in length. The string may be a literal or a variable or the result of string formulas, limited only to BASIC protocol. You would fix any SYNTAX errors within the parameter, the same way you would in a regular BASIC program.

A listing of both routines follows:

```
    GET'STRING JSR $AEFD
 JSR $AD9E
 JSR $B6A3
 LDX $22    ;useful for
 LDY $23    ;setnam
 RTS
GET'NUMBER JSR $AEFD
 JSR $AD8A
 JSR $B7F7
 RTS    ;LSB in . Y / MSB in . A
```

BUFFERS

Buffers are sometimes important to ML disk access. When processing one file while creating another, you might discover a significant increase in speed if you read a few blocks (block = bytes / 254) into a buffer and then process the data from your buffer until you've reached bottom. In layman's terms, your drive, even a hard drive or RAM drive, is more efficient at getting many bytes than a byte at a time. Generally a one-block buffer will generate a sufficient speed increase, but depending on your work, you may find that adjusting your buffer to a larger size provides optimum efficiency. Of course there is a ceiling where additional buffers offer no efficiency increase.

The smallest possible buffer is one byte, where you get a byte from one file, process it, and write it to another file. Unless you're working with lines in a file, I suggest skipping all sizes between 1 and 253. Start out with multiples of 254.

Managing buffers is simple:

1. Mark the beginning and end of the buffer.
2. Get bytes from your source file until buffer is full or file ends or there is a disk error.
3. Mark the highest byte written to in the buffer with a two byte integer (WORD). If the file ended, mark that separately.
4. Process the data as defined by your program.
5. Write processed data to the file you're creating.
6. If the file hasn't ended and there is no error, go back to step 2.

You can build a model of your buffer in BASIC, then translate the BASIC into ML. To make the translation easier, don't use arrays or strings for storage. Use memory locations (PEEKS and POKES). Use the cassette buffer or the $C000 area (49152-53247) for storage and retrieval. Don't use INPUT#. Use GET#n,a$. Think of a$ as your accumulator and use the variables x and y to index places in memory that you've designated for storage. For instance:

```
5 input"file";f$:input"device";d
10 open2,d,2,f$
20 y=0:buffer=49152:eof=0
30 get#2,a$
40 poke buffer+y,asc(a$+chr$(0))
50 y=(y+1)and255
60 if y <> 0 then 80
70 buffer = buffer + 256
80 if buffer = 53248 then 100
90 if st <> 0 then eof=1: goto 110
100 goto 30
110 close 2
120 final = buffer + y
130 end
```

This program will copy any file (up to 17 blocks) into a buffer that starts at $C000 and ends at $CFFF. it's not a fully implemented buffer routine since it only allows for one pass (it closes the file after the buffer is filled). It will be veeeeeeery slow because it's BASIC. Even if you expect it to be slow, it will still surprise you if the file you try it on is the full (whopping?) 17 blocks and it takes minutes to copy. Copying files is obviously more suited to ML.

The BASIC program will stop when the file reaches $d000 or if the file ends. Note that I used fall-thru logic in lines 60 and 90. This means that depending on the result of a test, the program would fall through without jumping over any instructions. Line 70 is executed only when y=0. Line 100 is executed only when the buffer <53248 and/or ST is <> 0.

Here's the ML equivalent, using GET'NUMBER and GET'STRING. The program would be called like this:

sys 32768,"text file",8

That SYS would open the file. This SYS will fill the buffer repeatedly until the file is closed.

sys32771

Copying 17 blocks into the buffer should take about 9 seconds without JiffyDOS, and under a second with it. On a HD or RAM disk, don't even bother timing it.

```
   org $8000
 buffer = 251
 jmp initialize
jmp fill'buffer
 initialize jsr get'string
jsr setnam
jsr get'number
tya
tax
LDA #2
tay
jsr setlfs
jsr open
jsr clrchn
ldx #2
```

The same thing is done for the text part of the routine. The computer is put into text mode, and the next (the third) interrupt call is set up.

MY FRIEND, FLICKER

This is a big sidebar by Jeff to make some VERY important points. Now in the previous programs we jumped to the OLD VECTOR routine when done. This is a very good idea normally. If the interrupt was not caused by the CIA timer, you should handle the entire interrupt yourself. Those routines expect to be called ROUGHLY every 1/60 of a second. The VIC chip and the TIMER both go at 60 hertz cycles, but since they are SEPARATE devices, they aren't exactly the same speed. The same as two identical quartz watches, physically identical, but constantly growing farther apart. Our timers aren't very accurate and will lose a total of a cycle in only a few seconds, not a year. When you notice your raster effect creeping down, flashing, etc. , the culprit is another interrupt interrupting your RASTER interrupt and restoring control to your raster interrupt after it's moved past the line that caused the raster interrupt.

Also if you have three interrupts on a screen and JMP (oldirq) after each raster, you're calling a routine THREE TIMES in a sixtieth of a second that was meant to be called only once in that time. TI$ will be updated too quickly. The cursor will flash too frequently. Quite contrary to how it appears, your system has NOT sped up.

On the other hand if you never JMP (oldirq) the keyboard won't work. SIDPLAYER won't work, etc. So you have to do a balancing act. Make sure that the IRQ doesn't infringe on your raster interrupt, but still JMP (oldirq) about 60 times a second. This means first of all that you must disable the timer interrupt:

```
LDA #$7f
STA $dc0d
```

Now the timer interrupt will NEVER be called until you do it manually. If the last raster line you work with is $55, try JMPing (oldirq) after that. The timer IRQ should be finished before your next raster interrupt. See how that affects flicker. Try pressing keys.

You can also try checking to see if it's time for an interrupt:

```
LDA $DC0D
and #1
bne timer'exit
 jmp $ea81
 timer'exit jmp (oldirq)
```

But never test for this in the middle of intensive raster interrupt activity like when you have only two lines before the next interrupt.

DO VISUAL STUFF **FIRST**

If you're changing fonts on line $50, your raster routine should change that font right away when the raster routine is called. If you're changing borders, same thing. Don't do math and setups. Get the visual stuff out of the way. Don't forget, the interrupt occurs when the raster hits your line, but it doesn't wait for you to finish your changes before it begins writing pixels. In the tightest routines, the raster has already begun writing just offscreen while you're still branching to your routine. You have precious MILLISECONDS to change the VIC chip before before the raster begins drawing visible pixels. Since the raster draws the border first, the border gets priority over the background. Since the top line of your font may be mostly blank, you might be able to afford setting the font last, but NEVER before something unnecessarily.

You can be doing everything right, but that annoying glitch is just because it takes you too long to set the font. For instance, you might try ANDing and ORing some number for the proper bank, screen and font location instead of just STAing the pre-determined number there. Why waste cycles with math when the result is going to be the same every time? If you have a font in a certain location in a certain bank with a screen in a certain area. Find out the hard numbers you're using.

I THINK YOU FORGOT SOMETHING

Only the IRQ interrupt has been explained so far (with no input on sprite interrupts or light pens/video pistols), but there are two other interrupts: the BRK and the NMI. BRK is triggered whenever the BRK command is executed. It doesn't quite fulfill our definition of an interrupt but it is similar. Normally, the BRK is vectored through locations $316-$317, which points to the same routine that is invoked by hitting STOP RESTORE.

NMI stands for non-maskable interrupts. This means it can't be turned off by the SEI command. The NMI can be triggered by either the RESTORE key or CIA #2. When triggered, it passes through the vector at $318-$319. This normally leads to a routine that determines the source of the NMI. If it was from CIA #2 then it jumps to some RS-232 routines. If it was the RESTORE key it checks to see if the STOP key is also pressed. Surely you know what this does! If not it simply returns.

File Buffers

By Jeffrey L. Jones. The lack of a file buffer has recently embarrassed me. I'll never make the mistake of writing a bufferless file copier again. You might notice a "just do it" attitude throughout the article. This is because disk access is just another task: You set it up, execute it, and you're done. It shouldn't be taken any more seriously than any other task. If you're familiar with disk access in BASIC, you'll find the transition to ML very easy. As mentioned last month, the most complicated part of ML disk access is opening the file.

TERMINOLOGY

For those who aren't familiar with MSB and LSB, they mean MOST SIGNIFICANT BYTE and LEAST SIGNIFICANT BYTE. These are the same as high bytes and low bytes, where two 8-bit numbers are combined to represent one 16-bit number with a range of 0-65535 where:

n=MSB*256+LSB.

You can verify this by doing the math for the number 2049, which has an MSB of 8 and an LSB of 1. Note the significance of the hexadecimal representation of 2049, $0801. The MSB and LSB are visible in hex, the MSB appearing first. Each hex digit represents a nybble.

KERNAL ROUTINES WE'LL USE

LABEL	HEX	DECIMAL
LOAD	$FFD5	65493
SAVE	$FFD8	65496
SETNAM	$FFBD	65469
SETLFS	$FFBA	65466
CHROUT	$FFD2	65490
PLOT	$FFF0	65520
CLRCHN	$FFCC	65484

your proper place in this community before attempting higher levels of access.
SysOp

Dear SysOp:
I kant figure out how to D/L [note user's discovery of BBS abbreviations]. I have tried to use XModem with no sucsess, and then I tried ZModem and everything went kablooie. Please help me so that I can download your good filez.
Joe Blow.

Dear Joe:
You're an idiot, but we like you. We think you have potential. I'll tell you what I'm going to do. I'm going to teach you how to use the file section, but you're going to learn to use it by UPLOADING [imagine teary-eyed face cringed with fear at this prospect]. Next time you log on, page me to chat and I'll show you how to upload a file.
The SysOp

Dear SysOp:
Okay.
Joe Blow

Transcript of first chat with SysOp follows:

Select [M, F, E, C, P, G]: P
Paging your SysOp! . . . . . .
The SysOp is here!
Hello, Joe!
NO CARRIER [Joe hangs up, torn with fear]

[Transcript of second chat with SysOp follows]

Select [M, F, E, C, P, G]: P

Paging your SysOp! . . . . . .
The SysOp is here!

Hello, Joe!
[Long pause]
Heelo.
Do you want me to show you how to upload, now?
[Another pause]
Yes.
Okay. . . I'll walk you through it.
Exiting chat . . .
Select [M, F, E, C, P, G]: F
Select [U, D, C, L, F, S]: C
Change to which area? 1
Changing to upload area (1).
Select [U, D, C, L, F, S]: U
Select a protocol
<X> XModem <Y> YModem <Z> ZModem
The SysOp is here!
Okay, Joe. Tell your program you want to upload by pressing PgUp when you want to start the transfer.

Exiting chat . . .
Select [X, Y, Z]: X
Enter filename to upload:
APROGRAM. ZIP

Begin your upload procedure. . .
1 file(s) transfered successfully!
The SysOp is here!
See, that wasn't so hard, was it?
Nnno.
Well, to download, you do the same thing in reverse.
Okay! Cool!
By the way, what was it you uploaded, anyway?
I don't know. . . I got it from one of my friends. It's something called a virus.
NO CARRIER

Dear Joe:
I looked at the program you uploaded. If you ever upload a virus again, I'll kill you slowly, and your little dog too. You have a lot to learn, kid.
The mildly pissed SysOp

PART II (one year later)

Dear SysOp:
I uploaded those files you asked me for. My upload ratio is now better than my download ratio. Can you pleez let me in on some of the better file areas?
Joe

Dear Joe:
Okay. I think you deserve it. I'm going to let you into some of the other file areas.
By the way, don't upload anything that has the words "cracked by" on it anymore. I could get in big trouble.
By the way, a protocol is a way to transfer files, not a matter of etiquette.
The SysOp

Dear SysOp:
Okay. I just thought you might like that game. Something about those Amazon Women just appeeled to me.
Joe

Dear Joe:
You skip school to call here, don't you? Can't you call somewhere else for a change?
The SysOp

Dear SysOp:
You mean there are other BBSes out there? Why didn't you tell me about them before?
Joe Blow

Dear Joe:
You never were this annoying before. Try the "Weirdo's Hideaway", 555-6543.
The SysOp

The user does not call for 7 weeks, as he discovers other BBSes. Eventually, he decides to call back and batch upload all the warez that he collected on his leech festivals

Dear SysOp:
I learned how to phreak last week! It's a lot of phun and you don't have to pay when you download philes [it is obvious that the kid has been calling California boards, where they spell all "f" sounds with "ph"]. Among the warez I'm uploading to you is a program called Code Thief. It will help you phreak too!
Joe

Dear Joe:
Phreaking is a bad business. Don't you think they can figure out where those calls you are making are coming from? They can. I suggest you stop before you get yourself and your parents in trouble.
Your SysOp

Dear SysOp:
What's wrong with phreaking? It's not like it's illegal or anything.
Joe.

Dear Joe:
You're so full of crap, you stink.
Your SysOp

Dear SysOp:
One of my friends that I met on a board in California [see! we told you] had something happen to him called "getting busted. " What does that mean?

```
jsr chkin
 check'drive jsr readst
and #128;is drive there?
sta eof
bmi quit;drive not online
 LDA <49152
sta buffer
LDA >49152
sta buffer+1
LDA #0
sta eof
rts
 fill'buffer ldy #0
loop jsr getin
sta (buffer),y
iny
bne over1
inc buffer+1
LDA buffer+1
cmp >53248;#d0
bcs quit
 over1 jsr readst
and #66
sta eof
bne quit
beq loop
 LDA buffer
sta final
sty final+1
 quit LDA eof
beq not'done'yet
 LDA #2
jsr close
not'done'yet jsr clrchn
rts
 get'string jsr $aefd
jsr $ad9e
jsr $b6a3
ldx $22
ldy $23
rts
 get'number jsr $aefd
jsr $ad8a
jsr $b7f7
rts
 final . word 0
eof . byte 0
```

Note that I checked for drive availability after CHKIN, not by closing the file as I would in BASIC. After CHKIN or CHKOUT, you can test for drive availability since these routines return errors through the accumulator and the ST variable.

Note that I ANDed the status with #66 instead of #64. This is because I'm not only interested in the end of file, but also errors. Bit 1 of ST reports on "time out read", which basically means that the drive attempted to read data, but a time limit had expired before the drive delivered. This can happen when there is any type of error where the drive can't read any longer. For instance, opening the drive door in the middle of reading. I could read the error channel after reading each byte, but that would be grossly inefficient and slow down disk access considerably.

ERROR CHANNEL

Something I neglected to mention last month was that the error message sent by the drive is a string -- all of it, even the error number. So don't go looking for LSBs and MSBs from the error channel. You're working with nothing but strings.

The error message will remain unchanged (usually "00,ok,00,00") unless there is a special message or a bona fide error message (trouble) after:

a. a disk command + carriage return
b. a read (other than error chan)
c. the entire error message is read in which case it reverts to "00,ok,00,00"
d. a write (other than error chan)
e. OPENing a file

It takes less code to test the error number as a string than to convert the string into an integer, then interpret the LSB and MSB. For instance, if you're checking for "50,record not present,00,00", you would get only two bytes from the error channel and process them in the following way:

```
 ;your code has just finished
;positioning the head in a
;RELative file. Now you want
;to see if the record exists.
;The file number for the error
;channel is 15
 jsr clrchn
ldx #15
jsr chkin
 jsr getin
cmp "5"
bne failed
jsr getin
cmp "0"
bne failed
 not'present LDA. . .
 ;your code continues below
```

The code, NOT'PRESENT, cannot be reached without passing two tests -- the test for the "5" and then the test for the "0". So the code must "fall through" to it. Note that the code, FAILED, should test and report on any other error. The code gives "50,record not present,00,00" special handling.

Also note: the error channel's message is placed in a FIFO (First In First Out) buffer. Once you GET two characters of the error message, it's up to you to preserve them. They don't exist in the drive's memory anymore. So if you go back to the error channel, the first byte you get will be "," then "r" instead of the error number.

If you have a buffer of about 40 characters, you can store the entire error message in the computer's memory, then you can test the error channel non-destructively.

```
jsr clrchn
 ldx #15
 jsr chkin
ldy #0
 loop jsr getin
 cmp #13;     carriage return
 beq done
 sta buffer,y
 iny
 bne loop
done sty error'length
 lda buffer
 cmp "0"
 bne error
 lda buffer+1
 cmp "0"
 bne error
     ;your code would continue here
```

Note that after this code is executed, the error channel is left in an "00, ok,00,00" state and the error message resides in your code's buffer. Now you can print this string at will.

A Musical Drama In -V- Parts

PART I

Note from Jeff. The author of this drama left his name to the winds of the net. My apologies if this toned-down drama is still too risqué. Despite the title, I doubt if you'll ever see this performed as an opera or musical play.

Dear SysOp:

I am a new uzer. I am ate yearz old and I have just gotten my first modem. I like to download lotsa files as long as you don't hafta upload in return. Pleez give me access on your bord.

Joe Blow.

Dear Joe:

I have given you minimal access in the kiddie file and message areas. We suggest that you learn to spell and learn

Available from LOADSTAR!

Chris Abbot's goal was to professionally reproduce well-loved Commodore demo and game tunes. He pulls this off quite well, using state-of-the-art MIDI equipment. These CDs were not manufactured on a PC's CD recorder. They were professionally pressed, fully packaged and contain a nice little booklet with explanations for each song along with a Rob Hubbard interview. You should get this CD, if only as a collector's item. The item number is #200122

LOADSTAR 1-800-594-3370

charges of beating a seven-year old girl, shooting the kid, and killing a little dog. He was sentenced to die in the electric chair, but went with a big grin on his face.

The kid went to hell, where all little leeches eventually go. His sister went to hell too, and sold shoehorns for a living.

THE MORAL

SysOps: THIS COULD HAPPEN TO YOU! Don't let nine-year-olds on your BBSes! Ban the little leeches!

Note from Jeff: When I was a sysop, a nine year old little boy gave me the most trouble. His drive to gain access to the adult areas was unparalleled in all the animal kingdom. Of the hundreds who logged on, only he logged on under assumed names and only he phreaked out the account of a nice young lady who happened to be the daughter of a friend of mine. He then logged on and used her name to post embarrassing messages. He was the only real problem that I had.

What Men Really mean

☺ **"I'm going fishing."** Really means. . . "I'm going to drink myself dangerously stupid, and stand by a stream with a stick in my hand, while the fish swim by in complete safety."

☺ **"It's a guy thing."** Really means. . . . "There is no rational thought pattern connected with it, and you have no chance at all of making it logical."

☺ **"Can I help with dinner?"** Really means. . . . "Why isn't it already on the table?"

☺ **"Uh huh," "Sure, honey," or "Yes, dear."** Really means. . . . Absolutely nothing. It's a conditioned response.

☺ **"It would take too long to explain."** Really means. . . "I have no idea how it works.

☺ **"We're going to be late."** Really means. . . . "Now I have a legitimate excuse to drive like a maniac."

☺ **"I was listening to you. It's just that I have things on my mind."** Really means. . . . "I was wondering if that red-head over there is wearing a bra."

☺ **"Take a break, honey, you're working too hard."** Really means. . . . "I can't hear the game over the vacuum cleaner."

☺ **"That's interesting, dear."** Really means. . . . "Are you still talking?"

☺ **It's a really good movie."** Really means. . . . "It's got guns, knives, fast cars, and beautiful women."

☺ **"That's women's work."** Really means. . . . "It's difficult, dirty, and thankless."

☺ **"You know how bad my memory is."** Really means. . . . "I remember the theme song to 'F Troop', the address of the first girl I ever kissed and the Vehicle Identification Numbers of every car I've ever owned, but I forgot your birthday."

☺ **"I was just thinking about you, and got you these roses."** Really means. . . . "The girl selling them on the corner was a real babe."

☺ **"Oh, don't fuss. I just cut myself, it's no big deal."** Really means. . . . "I have actually severed a limb, but will bleed to death before I admit I'm hurt."

☺ **"Hey, I've got my reasons for what I'm doing."** Really means. . . . "And I sure hope I think of some pretty soon."

☺ **"I can't find it."** Really means. . . . "It didn't fall into my outstretched hands, so I'm completely clueless."

☺ **"What did I do this time?"** Really means. . . . "What did you catch me at?"

☺ **"I heard you."** Really means. . . . "I haven't the foggiest clue what you just said and I'm hoping desperately that I can fake it well enough so that you don't spend the next 3 days yelling at me."

☺ **"You know I could never love anyone else."** Really means. . . . "I am used to the way you yell at me, and realize it could be worse."

☺ **"You look terrific."** Really means. . . . "Oh, God, please don't try on one more outfit. I'm starving."

☺ **"I'm not lost. I know exactly where we are."** Really means. . . . "No one will ever see us alive again."

☺ **"We share the housework."** Really means. . . . "I make the messes, she cleans them up."

Twins!

An unmarried woman is newly pregnant and gets into an auto accident. She suffers a head injury and lapses into a coma for nine months. When she awakens in the hospital, she panics and asks about her baby.

Her doctor is called in and gives her a mild sedative, then he sits down to answer her questions. "I'm so happy to see you recovering", he says. The woman responds, "Thank you doctor, but what about my baby? Is everything all right?" He replies, "Yes, despite your injury, we were able to perform a fairly normal delivery procedure."

"In fact," he goes on, "you've given birth to twins - a boy and a girl."

The woman is very happy and asks when she can see her new babies. The doctor replies, "Right away, but we've already sent the infants home with your brother. We'll call and tell him you're okay. While you were unconscious, your brother took care of everything for you. He even gave the babies names."

At this point, the woman gets upset, "Doctor, my brother is an idiot! What name did he give my little girl?" The doctor answered that her name was Denise. "Oh, Denise, that's not so bad. What name did he give my boy?" The doctor answered, "Denephew".

LOADSTAR LETTER #62

J&F PUBLISHING 606 COMMON STREET SHREVEPORT LA 71101

- COMMODORE NEWS
- COMMODORE VIEWS

Bulk Rate
U.S Postage PAID
Shreveport LA
PERMIT #85